|  |
| --- |
| [Nom de la société] |
| [Titre du document] |
| [Sous-titre du document] |

|  |
| --- |
| Julien Rochat  [Date] |

Table des matières

[1 L’énoncé 1](#_Toc484597808)

[1.1 Descriptif 1](#_Toc484597809)

[1.2 Travail demandé 1](#_Toc484597810)

[2 Le résumé 2](#_Toc484597811)

[3 Avant-propos 3](#_Toc484597812)

[4 Réseau de neurones 4](#_Toc484597813)

[4.1 Introduction 4](#_Toc484597814)

[4.1.1 Historique 4](#_Toc484597815)

[4.2 Réseau de neurones classiques 4](#_Toc484597816)

[a. Introduction 4](#_Toc484597817)

[4.2.1 Fonctionnement 4](#_Toc484597818)

[4.2.2 Activation 5](#_Toc484597819)

[4.2.3 Quelques exemples de fonctions d’activations : 6](#_Toc484597820)

[4.2.4 Apprentissage 6](#_Toc484597821)

[4.3 Réseau de neurones convolutifs 7](#_Toc484597822)

[4.3.1 Introduction 7](#_Toc484597823)

[4.3.2 Convolution 7](#_Toc484597824)

[4.3.3 Couche 9](#_Toc484597825)

[4.3.3.1 Couche convolution 9](#_Toc484597826)

[4.3.3.2 Couche pooling 9](#_Toc484597827)

[4.3.3.3 Couche correction 9](#_Toc484597828)

[4.3.3.4 Couche entièrement connectées 10](#_Toc484597829)

[4.3.3.5 Couche perte 10](#_Toc484597830)

[4.3.4 Exemples 10](#_Toc484597831)

[5 Techniques 11](#_Toc484597832)

[5.1 Introduction 11](#_Toc484597833)

[5.2 Analyse fonctionnel 11](#_Toc484597834)

[5.2.1 Spécification 11](#_Toc484597835)

[5.2.2 Analyse de l’existant 12](#_Toc484597836)

[5.2.2.1 Déploiement 12](#_Toc484597837)

[5.2.2.2 Communication 13](#_Toc484597838)

[5.2.2.3 Application 13](#_Toc484597839)

[5.2.2.4 Réseau de neurones 14](#_Toc484597840)

[5.2.3 Choix technologiques 15](#_Toc484597841)

[5.2.4 Framework 15](#_Toc484597842)

[5.2.4.1 Symfony 15](#_Toc484597843)

[5.2.4.2 Docker 17](#_Toc484597844)

[5.2.4.3 Websocket 18](#_Toc484597845)

[5.2.5 Lasagne 19](#_Toc484597846)

[5.3 Analyse organique 20](#_Toc484597847)

[5.3.1 Architectures 20](#_Toc484597848)

[5.3.2 Base de données 21](#_Toc484597849)

[5.3.3 Serveur 22](#_Toc484597850)

[5.3.3.1 Les fichiers 22](#_Toc484597851)

[5.3.3.2 Les routes 23](#_Toc484597852)

[5.3.3.3 Websocket 24](#_Toc484597853)

[5.3.3.4 Fonctionnement 25](#_Toc484597854)

[5.3.4 Client 26](#_Toc484597855)

[5.3.4.1 Python 26](#_Toc484597856)

[5.3.4.2 Web 26](#_Toc484597857)

[5.4 Tests et performances 26](#_Toc484597858)

[5.4.1 Tests 26](#_Toc484597859)

[5.4.2 Performances 26](#_Toc484597860)

[5.5 Déploiement 26](#_Toc484597861)

[5.6 Installations 26](#_Toc484597862)

[5.7 Amélioration 26](#_Toc484597863)

[6 Expérience 27](#_Toc484597864)

[6.1 Introduction 27](#_Toc484597865)

[7 Annexes 28](#_Toc484597866)

# L’énoncé

## Descriptif

Ces cinq dernières années les réseaux de neurones artificiels convolutionnels (CNN) ont connu un grand essor dans un certain nombre d'applications d'intelligence artificielle comme la reconnaissance d'objets. La première partie de ce travail d'initiation dans ce domaine consiste à plonger dans l'univers des réseaux de neurones artificiels pour lequel peu de cours ont été donnés dans le cursus scolaire de hepia. L'étudiant devra faire preuve d'autonomie pour la compréhension de systèmes d'apprentissage statistique standard et profond. Dans la deuxième partie il sera question d’implanter un logiciel permettant de définir et d’exécuter facilement l’apprentissage de réseaux CNN sur des machines disposant de GPU. De cette manière le temps d’apprentissage sera significativement raccourci. Quelques cas pratiques d’application seront explorés, comme la reconnaissance d’objets.

## Travail demandé

* Comprendre les modèles de réseaux de neurones artificiels standards - Comprendre les modèles de réseaux de neurones artificiels convolutionnels
* Comprendre et utiliser au moins un «framework » pour l’utilisation de CNN - Utilisation de CNN pour la mise au point d’une application
* Compréhension de l’utilisation de GPU pour l’apprentissage de CNN - Implantation d’un module facilitant l’utilisation du GPU pour les CNN
* Réalisation d’un ou plusieurs exemples d’apprentissage par GPU - Analyse des résultats - Rédaction du rapport

# Le résumé

# Avant-propos

# Réseau de neurones

## Introduction

Dans la première partie de mon travail de bachelor, je vais expliquer le fonctionnement général des réseaux de neurones et ensuite je vais détailler le fonctionnement d’un réseau de neurones convolutifs.

### Historique

|  |  |  |
| --- | --- | --- |
| Dates | Auteurs | Apport |
| 1890 | W. James | Concept de mémoire associative – Loi de fonctionnement pour l’apprentissage |
| 1943 | W. McCulloch et W. Pitts | Modalisation du neurone biologique en neurone formel |
| 1957 | F. Rosenblatt | Modèle du perceptron |
| 1969 | M. Minsiky et S. Papert | Mise en avant des limites du Perceptron |
| 1980 | Kunihiko Fukushima | Création du modèle Neocognitron |
| 1985 | Rumelhart et Y. LeCun | Réseaux de neurones multicouches – rétro propagation du gradient |

## Réseau de neurones classiques

### Introduction

Dans ce chapitre, je vais parler des réseaux de neurones en général et dégrossir les différentes propriétés grâce à des exemples. Les réseaux de neurones en informatique sont basés sur le fonctionnement des neurones biologiques. Généralement, des méthodes d’apprentissage sont utilisées pour apprendre aux réseaux de neurones à traiter des données.

### Fonctionnement

Le principe du fonctionnement d’un réseau de neurones est de présenter des données à la première couche. Cette dernière va effectuer un traitement sur les différentes données et ainsi activer différents neurones qui seront traités de la même façon à la couche suivante jusqu’à arriver à la dernière couche qui correspond à nos différentes sorties possibles en fonction du problème donné. Par exemple, j’ai des données différentes concernant des animaux (Taille, Couleur, Vertébrés …), le réseau de neurones va alors traiter ces diverses informations et déterminer à quel animal correspondent le plus ces données dans la couche de sortie.

Pour fonctionner, un neurone doit avoir des entrées qui correspondent aux neurones de la couche précédente ou directement aux données de la première couche. Chaque entrée est associée à un poids. Le principe est d’utiliser le poids pour multiplier les différentes valeurs d’entrées et faire la somme des diverses valeurs obtenues pour chaque entrée. Ensuite, le résultat passe à travers une fonction d’activation qui permet de déterminer la sortie de ce neurone.

### Activation

La fonction d’activation est une fonction mathématique qui permet de déterminer un seuil d’action pour la valeur de sortie d’un neurone. Elle est généralement non linéaire ce qui permet d’approximer n’importe quelle autre fonction. On peut la représenter en trois étapes pour se faire une idée de son fonctionnement. Quand la valeur présentée est en dessous du seuil d’activation, la sortie va généralement donner des valeurs comme 0 ou -1. Quand la valeur est proche du seuil on va avoir une phase de transition et quand la valeur est au-dessus du seuil on va avoir une valeur qui vaut 1. Les valeurs présentées peuvent varier en fonction des différentes fonctions d’activations.

### Quelques exemples de fonctions d’activations :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Graphe** | **Équation** |
| Identité | https://lh3.googleusercontent.com/csyqs2mi_xEbetIntPpUOA1XVcnXyr_9jVd8NYoX2tnPDr_dHDaoddCInC-MnHzE8KfVdFQqeVqfvAWpnVZjB_DOT5ZMT2HLy3Zd4fkgoYbDVlVy96DGpbpJ_JQJaC2ssiArX0-A | https://lh4.googleusercontent.com/cTofGO59b9Asg03LifhdbPBqoCeH0ZlFjg2k6CIajWiAkiXZ21RtJJRbNHjCDN1Isttq1Ywk1fxoEK1tnNg5R96cDaBtRYgE-LZn6KNXxEa7042F7f8AOX7RYGqPVyUa6YSP2jBB |
| Marche/Heaviside | Activation binary step.svg | https://lh4.googleusercontent.com/ShP0TenjQYwKzYDcFBlPPSJUmKM1pKCo57syjXS-ZCh-vnE7JEDCtsCvNOUQCjsfFRi4bJsDm17PTpv5hnBT-saRpJH2kvAXB33Avuh7pSe4mSnvYO3n8_tY8dV1RuUh0tYQcoG4 |
| Logistique (ou marche douce) | https://lh4.googleusercontent.com/_vYrO6_w_q33jPTmJ8t0PM0-uKRks4DPtE_aeXjZguajQIVFy-5zU83Er4T0n5EaWOldoHAOr7HVso_f5tKOPuOHFtzGX3UghTQSKOGwgnW6or2dSNq_BAs5wv6J41g04NgAPo38 | https://lh4.googleusercontent.com/YhbMXW8GCrAJruDHSbesZ0IY0WRRTn9xhLE3qtI1tsJZDpNCdZ3YOPGunx_Gu4-ZEIW7Ac1udw4b2KrM7d72l3vnc6LhWO4tguPG3Rn_6Hf1ZlbM7J50I-cFpgLwKEn0CQff7sxM |
| Tangente Hyperbolique (TanH) | https://lh4.googleusercontent.com/4lQlVDAbJixFLIXzwsiY86YgXIL4cudbYdoQX8J8yU-FQ2D9pCNEYe508Yv-2JfKrgt8z4godfC4IU5JjolV1rfEIJ7FQ4a8MQSBzmb-RG7a6-60kotr7FQWUAxkgdfoPHTrycMe | https://lh4.googleusercontent.com/V3YOJmAsA_7lgyt-s7oF5x_1AFwpOzpHUxzf9gg0791SQYWxzcqRDg_0U_r88efdEbBqa6c97I8w5U3lTrYP-qChfP7W8-wrxweWV8j_vGB-OJo-8kUSWV5oT7YkR_zKzYZWYMOl |
| Arc Tangente (ArcTan ou Tan-1) | https://lh6.googleusercontent.com/ubU9frW2Vz_VQJC4W2y_cK4-6D8gVgWQN79X8uhnIhopqGO77ss7YuFubgRGVeVQfn7qYZvNV1DuPKtElcuI0WL7OeQsULfB8faubiSJSNdkm-6m8od8vkwlsWO8qBDPvf5KOzcc | https://lh4.googleusercontent.com/ylCnAYtammepaW6e9nIJp7_56kzCnolRnazHI_aTwd4zxWJGpBTtA4rA4HPchRWfkwdQU_-QWX44TDDCZIkSIu-GLPGZaO_OVk6-V3KQJJtDayynbzvvkpbF-7V-7dcPK7EW9hlf |
| Signe doux | https://lh5.googleusercontent.com/tFBzFW-j7JDiAnLiM4v8SONxPOOImcrEMF85GW2gLN07dKaR4hjgxIGesPpaoZQAwIMe67nb7OoYbqI0bYolbkFiIFGDuviPlOZbMKgtaPOch7mQZnXP3JbTyF7_otyBatdgCp8h | https://lh3.googleusercontent.com/6VM1-qXr_iGT7UNxc0qtgvPirEte2ZbMIq-REh5MEkoMK_yTPGg2wYbPrBID26PYFnaK8Q05bpPUQMcHD1RweuQgnenIn7uuOtQmOnBnnww08QpwR61lgzhqQnMTReoaUaS1J6ak |
| Unité de Rectification Linéaire (ReLU) | https://lh5.googleusercontent.com/hgNn_Oz5JSWHTDO2c8sabMKvML8TwBnD-eyMJ-NyS76I4kqgfuPN2BF8x38B-YRc1LBnwBa0ipYDsjJfCdavoNdeYwlYiZj5wcQfgynMQdLOVKpOWyVo_fGkrxUzyD7Zl0AOZYd5 | https://lh6.googleusercontent.com/wsb2baBC_7i7o2kaPvAR4mUEaSc-ul4oCC-9bGHmFzQg-RZh-4UXzGDMrpwKCWcWImGxKulFAFaNchk89Rcc6r1eke0izIP3hx-YMHnSkD0enwe_LL_ccWqFiIm9sIa9EUHJSn_C |

### Apprentissage

L’idée derrière l'apprentissage des réseaux de neurones est de présenter suffisamment de données en fonction du problème traité et ainsi de faire varier les poids du réseau. Cela va entraîner une mémorisation des différents exemples et ensuite permettre de généraliser ces données pour pouvoir déterminer des cas encore non rencontrés, mais similaires.

Il existe deux méthodes d’apprentissage : La méthode supervisée et non supervisée. Concernant la méthode d’apprentissage supervisée, on présente nos exemples et on aimerait qu’ils convergent vers un état final souhaité. Pour la méthode non supervisée on ne fait que présenter les exemples et on le laisse converger vers n’importe quel état final.

Pour apprendre, le réseau a besoin de modifier ses poids. On va donc modifier ses poids en fonction des erreurs commises pendant l'apprentissage d’un exemple.  Donc, si je présente des données qui correspondent à un singe et que le réseau de neurones trouve un dauphin à la sortie il va adapter les poids qui ont mené à cette erreur grâce à la rétrogradation du gradient de l’erreur. Ce qui permet de modifier les poids qui ont conduit à l’erreur plus significativement que les poids qui ont engendré une erreur marginale.

## Réseau de neurones convolutifs

### Introduction

Dans ce chapitre, je vais détailler le fonctionnement des réseaux de neurones convolutifs. Les réseaux de neurones convolutifs ou CNN sont inspirés par le cortex visuel des animaux. Ils sont utilisés dans des domaines comme la reconnaissance d’image et vidéo. Les CNNs sont composés de plusieurs couches :

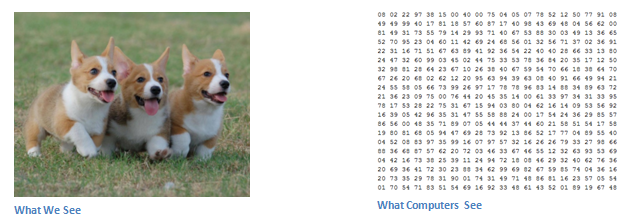
* Couche de convolution
* Couche de pooling
* Couche de correction
* Couche entièrement connectée
* Couche de perte

Ces différentes couches sont inspirées par les processus biologiques constatés dans les régions du champ visuel. L'idée derrière les CNNs est d'utiliser les poids des neurones vus précédemment pour apprendre des filtres. Les filtres permettent de mettre en avant des formes dans les images grâces aux différents exemples appris. Ce qui va permettre de reconnaître l’objet en question en fonction des différents filtres qui se sont activés.

### Convolution

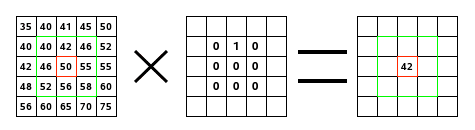
Pour commencer, nous allons parler de la convolution qui est la partie la plus importante d’un réseau convolutif.

En informatique, nous stockons les images par des nombres qui représentent la valeur des pixels de l’image. Par conséquent, nous pouvons considérer notre image comme une matrice.



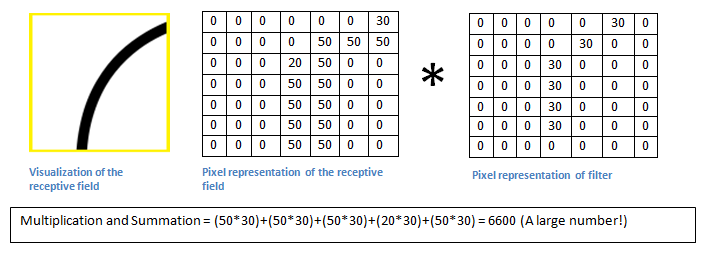
La convolution permet d’effectuer le filtrage d’une image grâce au traitement de la matrice de notre image par une autre matrice qui est appelée matrice de convolution ou noyau.

De ce fait, la matrice de convolution va faire office de filtre, elle va étudier chaque pixel de l’image en multipliant et en additionnant les deux matrices.

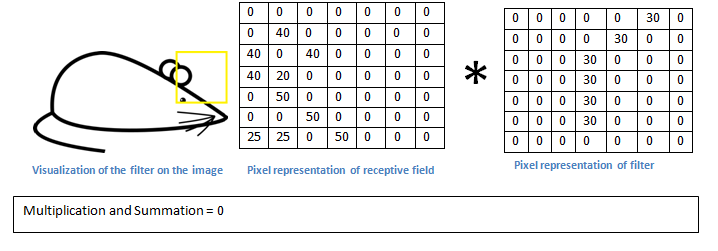


(40\*0)+(42\*1)+(46\*0) + (46\*0)+(50\*0)+(55\*0) + (52\*0)+(56\*0)+(58\*0) = 42

Dans le cas d’un réseau de neurones convolutifs nous allons par exemple avoir un filtre qui va déterminer si l’image correspond à une courbe.



On peut voir dans l’image ci-dessus que la partie de gauche correspond à la partie d’une image et la partie de droite correspond à un filtre. On veut déterminer si cette image ressemble à une courbe et on utilise à cette fin la formule de la convolution. Ce qui nous donne 6600 car l’image ressemble beaucoup à notre filtre.



Maintenant on prend une partie de l’image qui ne ressemble pas à notre filtre et on répète l’opération de convolution. On remarque que cette fois le résultat est 0.

On peut en conclure que plus le résultat de la convolution est élevé plus la partie de l’image ressemble à notre filtre.

A la fin de l’application du filtre sur chaque pixel de l’image on va se retrouver avec une nouvelle image qui va mettre en avant les différentes courbes que comportait l’ancienne image. Ce qui révèle un des points forts de la convolution c’est qu’il permet de détecter un élément partout dans l’image et différé de l’original.

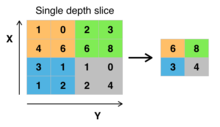
### Couche

#### Couche convolution

La couche de convolution utilise le principe de convolution vu avant pour apprendre des filtres aux différents poids des neurones. Les poids sont partagés dans une couche de convolution car on considère que si le filtre est efficace sur une partie de l’image il l’est aussi pour le reste de l’image.

#### Couche pooling

La couche de pooling (mise en commun) permet de redimensionner le réseau entre différentes couches. La couche de pooling va découper l’image d’entrée pour échantillonner l’image en plusieurs petites parties.



Par exemple, dans l’image on voit que l’on garde seulement les maximums ce qui permet de réduire la taille de l’image et ainsi d’appliquer des filtres plus précis sur les sous parties. Il existe d’autres fonctions de pooling par exemple l’average pooling qui permet de garder la moyenne de la partie découpée. Mais le max pooling est plus efficace car il augmente plus significativement l’importance des activations fortes.

Cette couche permet aussi d'économiser en performance car on aura moins de données à traiter par la suite.

#### Couche correction

La couche de correction est une fonction d’activation vue précédemment dans le chapitre des réseaux de neurones. Elle va permettre d’augmenter les propriétés non linéaires de l’ensemble du réseau de neurones.

Quelques exemples de fonctions d’activations utilisées dans les CNNs.

La fonction ReLU: https://lh5.googleusercontent.com/XQP02l0qC9DQclz2aSv5yG_AZzXALh-h6u2m7mJs5rqwShwi1Hg4gZArZwi3nCFvRrSYV1ynDGtIiK2Zxg3MD1_qDw6-JIMQ5s-joM2JfaJv-uZnR2awXUs5jpiloYcwz7AlKhQv

La fonction tangente hyperbolique : https://lh3.googleusercontent.com/0ziNesEnWPaitxycfA9TSs_zGva8FsAeCHnPeo3KnmXi333wJ7ech6bESroay-ruFa-VsSjcvyE_B20Q1cc_p72pc68CBUcCivjSsitjrxVwRSnY7IqeZRa7yPN88xMIyR4aCJJQ

La fonction tangente hyperbolique saturante :   https://lh5.googleusercontent.com/Rox8VrJKgD0GmlcX1GTrqsGP_oyfImqPvGRhCUY1_Z5oT_-1QIqO0RpagaT498SkruM5wMDRESYIn0NHs1wAiG1ce_aaTOvOQ0mZDvllrZRuOAo3B9jyqMYtC_DdM66LYd9lFiRQ

La fonction sigmoïde : https://lh4.googleusercontent.com/0m1v0jRXGis084wx3pjMa3ErPnf83v2VUvTxA1B2EyfDvxDC0WPrMDo9kvGOmbN03nPF7_eRbJq3bpBGxkqNp9L1pXRzYOPiy31CSL00akmrI_mRVEbb6D3Qf5LXT-yvXjLvBLaW

Généralement, on utilise la fonction de ReLU car la formation de réseaux neuronales est plus rapide.

#### Couche entièrement connectées

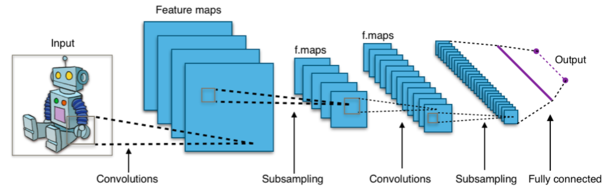
La couche entièrement connectée est la couche finale de notre CNN, elle permet d’appliquer le raisonnement haut niveau. Les neurones dans la couche entièrement connectée ont des connexions vers toutes les sorties de la couche antérieure. Ce sont ces neurones qui vont nous permettre de déterminer quelle classe va être activée.

#### Couche perte

La couche de perte permet de spécifier comment l'entraînement modifie le réseau entre le signal prévu et réel. Différentes fonctions sont utilisées en fonction des besoins. Par exemple, la perte softmax est utilisée pour prédire une seule classe parmi K classes mutuellement exclusives.

### Exemples

Malheureusement, il n’existe pas de formule magique pour trouver l’architecture d’un réseau de neurones convolutifs. Mais voici quelques exemples qui sont généralement utilisés :

* INPUT -> FC
* INPUT -> CONV -> ReLU -> FC
* INPUT -> [CONV -> ReLU -> POOL] \* 2 -> FC -> ReLU -> FC
* INPUT -> [CONV -> ReLU -> CONV -> ReLU -> POOL] \* 3 -> [FC -> ReLU] \* 2 -> FC
* 

# Techniques

## Introduction

Dans la partie technique de mon travail de bachelor je dois réaliser un service de déploiement qui va permettre de déployer rapidement des réseaux de neurones sur différentes machines via une application graphique.

## Analyse fonctionnelle

### Spécifications

L’utilisateur de l’application doit pouvoir nommer et décrire une architecture mise à disposition sur le serveur. Une architecture correspond aux différentes couches d’un réseau de neurones.

L’utilisateur de l’application doit pouvoir nommer et décrire un ensemble de données. Un ensemble de données correspond aux entrées et sorties d’un réseau de neurones.

L’utilisateur de l’application doit pouvoir créer un projet. Un projet permet de regrouper les différentes instances liées à la même problématique.

L’utilisateur de l’application doit pouvoir lancer des instances. Une instance est un entrainement d’une architecture avec un ensemble de données spécifiques sur un des clients.

L’utilisateur de l’application doit pouvoir visualiser les résultats de l’entrainement sous la forme d’un graphique.

L’utilisateur de l’application doit pouvoir stopper, mettre en pause, démarrer, reprendre et sauvegarder une instance.

L’utilisateur de l’application web doit pouvoir comparer les résultats entres les différentes instances via des graphiques.

L’utilisateur de l’application a la possibilité de lancer une validation croisée en sélectionnant plusieurs ordinateurs lors de la création d’une instance.

Le serveur doit pouvoir stocker et afficher les différents fichiers concernant les architectures et ensemble de données.

Le serveur doit contenir un système de validation croisée pour les ensembles de données. La validation croisée permet de séparer notre ensemble de données en dix parties, chaque partie contient un ensemble d’apprentissage et un ensemble de validation différent. Par exemple, pour 10 000 données il y aura 1000 données consacrées à la validation et 9000 données consacrées à l’apprentissage. Les données consacrées à la validation et à l’apprentissage seront différentes dans chaque partie jusqu’à avoir fait un tour complet des possibilités. Ce système permet de validité la fiabilité d’une architecture grâce à la moyenne des différents ensembles.

Le serveur doit pouvoir gérer les différentes instances disponibles.

Le serveur doit pouvoir envoyer des ordres aux différentes instances sur les clients pythons.

Le client doit pouvoir recevoir des ordres de la part du serveur.

Le client doit pouvoir gérer une instance en fonction des différents ordres que l’utilisateur souhaite exécuter.

Le client doit pouvoir télécharger les différents fichiers nécessaires au fonctionnement de l’entrainement d’un réseau de neurones.

Le client doit pouvoir interpréter l’architecture déterminée par l’utilisateur.

### Analyse de l’existant

#### Déploiement

##### Docker

Docker est un logiciel libre qui permet d'automatiser le déploiement d'applications dans des conteneurs. Ce qui permet d’exécuter des processus de façon isolée.

**Avantages :**

1. Déploiement rapide de plusieurs clients sur une machine.
2. Facilement portable d'une machine à une autre.
3. Automatisation de l'installation des différents éléments.
4. Léger en taille et rapide d’exécution.

**Désavantages** :

1. Un peu plus compliqué d'accéder au GPU.
2. Les conteneurs Docker possèdent par défaut la même adresse mac.

##### Machine virtuelle

Une simple machine virtuelle avec VMWare ou bien VirtualBox.

**Avantages :**

1. Déploiement rapide de plusieurs clients sur une machine.
2. Facilement portable d'une machine à une autre.
3. Automatisation de l'installation des différents éléments.

**Désavantages** :

1. Très compliqué et accès au GPU lent
2. Les images sont beaucoup plus lourdes car elles contiennent le système d'exploitation en entier.

##### Installation manuelle

L'installation manuelle ou via script des différents composants nécessaires au bon fonctionnement du projet.

**Avantages :**

1. Facile d'accéder au GPU.
2. Léger en taille et rapide d’exécution.

**Désavantages** :

1. Difficilement portable d'une machine à une autre.
2. Pas d'isolation entre les différents clients de la même machine.
3. Nécessite les droits administrateurs.

#### Communication

##### WebSocket

WebSocket est une spécification d'un protocole qui permet la communication bidirectionnelle sur un socket TCP entre un client et un serveur.

**Avantages :**

1. Permet d'avoir une communication full duplex pour le web.
2. Augmente l'efficacité entre la communication client-serveur.

**Désavantages** :

1. Obligation d'utiliser le protocole WebSocket donc d’avoir une librairie pour tous les clients qui désirent se connecter au serveur.

#### Application

##### Symfony

Symfony est un framework PHP pour le développement web. Il permet d'utiliser les bonnes pratiques de développement PHP pour concevoir des sites web de qualité notamment grâce à ses outils qui permettent de mettre en place les environnements de test et de production, de concevoir les contrôleurs et les templates sans oublier d'automatiser la gestion de la base de données.

**Avantages :**

1. Gain de temps en réutilisant des composants déjà existants.
2. Extensible et modulaire.
3. Système de template.
4. Une communauté active et une bonne documentation.

**Désavantages** :

1. Le temps d'apprentissage pour maîtriser correctement le *Framework* et les bonnes pratiques.

##### Angular2

Angular2 est un Framework JavaScript qui est fondé sur l'extension du langage HTML par de nouvelles balises et attributs pour aboutir à une définition déclarative des pages web.

**Avantages :**

1. Amélioration de la réactivité de l'interface utilisateur.
2. Intégration des tests.

**Désavantages** :

1. Long et difficile à apprendre.
2. Difficile de gérer une base de données.

##### Ionic

Ionic est un mélange d'outils qui permet de développer des applications web, mobiles et desktops. Il est basé sur AngularJs et Cordova. Ce qui permet de déployer une application sur plusieurs environnements.

**Avantages :**

1. Permet de déployer l'application facilement sur plusieurs plates formes.

**Désavantages** :

1. Reprend les mêmes désavantages qu’Angular2.

#### Réseau de neurones

##### Lasagne

1. Lasagne est un framework Python qui permet de créer et d'entraîner des réseaux de neurones dans Theano.

**Avantages :**

1. Il supporte plusieurs architectures de réseaux de neurones dont le réseau de neurone convolutif.
2. Il est écrit en python ce qui permet d’écrire rapidement du code et c’est aussi un langage que je maîtrise bien.
3. Il s’occupe de gérer l’optimisation des performances pendant l'entraînement des données.

**Désavantages** :

1. Il peut utiliser seulement une carte GPU pendant l'apprentissage.
2. Un peu plus difficile à prendre en main.

##### Caffe

Caffe est un framework d'apprentissage profond créé pour être modulaire et rapide.

**Avantages :**

1. Le code est extensible et le développement est toujours actif.
2. Il est performant dans l'apprentissage des données.
3. Il y a une grande communauté derrière ce qui permet de trouver facilement de l’aide.
4. Permet de déployer des architectures via des fichiers de configuration.

**Désavantages** :

1. Le code est difficile à comprendre.
2. Difficile d'intégrer une connexion réseau.

### Choix technologiques

Concernant la partie déploiement, j'ai décidé d'utiliser Docker car comparé aux machines virtuelles, il ne possède pas un système d'exploitation et il peut accéder directement au périphérique sans les émuler, ce qui permet de gagner en place et en performance. Par rapport à une installation manuelle les performances sont légèrement inférieures mais en contrepartie on gagne en mobilité et dans le cadre de mon projet de bachelor j'ai besoin de déployer rapidement et indépendamment du système d'exploitation sur un grand nombre de machines. Docker permet aussi de démarrer plusieurs instances isolées sur la même machine ce qui permet d'avoir facilement plusieurs clients qui tournent en même temps.

Pour la communication j'ai proposé seulement un choix technologique car pour la partie client j'utilise forcément une application web du coup WebSocket est la meilleure solution possible pour la communication client-serveur.

A propos de l'application j'ai décidé de partir sur le Framework Symfony, il est plus simple à apprendre qu’Angular2 et Ionic. Et il comporte tous les éléments nécessaires pour le bon développement d'une application MVC, la gestion automatisée de la base de données a aussi motivé mon choix.

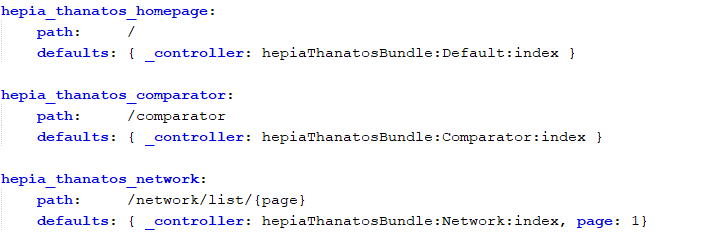
Concernant le réseau de neurones j'ai décidé d'utiliser Lasagne malgré le fait que Caffe comporte déjà un système de fichier de configuration pour l'architecture. Le code de Lasagne est plus compréhensible et va faciliter l'intégration des communications réseaux. De plus je maitrise mieux python que C++.

### Framework

#### Symfony

##### Routeur

Le routeur dans Symfony permet de déterminer quel contrôleur appeler et avec quels arguments. Ce qui permet d'avoir des urls personnalisées en fonction de la page. Par exemple, /comparator permet d'accéder à la page de comparaison de graphique. Les urls sont ainsi plus simple à retenir pour les utilisateurs.



Le code ci-dessus possède trois blocs distincts. Chacun correspond à une route. Une route est à chaque fois composée d'une entrée (path) et d'une sortie.

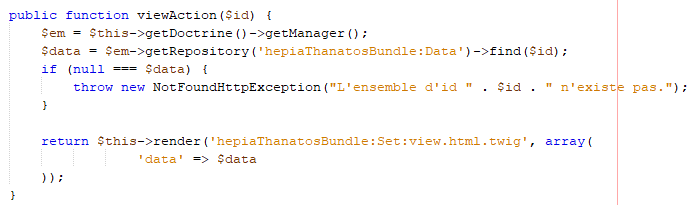
Quand l'utilisateur demande une url au site web le routeur va parcourir la liste des urls disponibles jusqu’à trouver une url qui correspond à sa demande et ainsi faire correspondre le chemin à un contrôleur.

Analysons en détail le troisième bloc, on peut voir que le chemin correspondant à la liste des réseaux et {page} nous permet de définir un paramètre variable. Dans notre cas {page} correspond à notre numéro de page. Dans default on peut voir qu'on fait appelle à notre contrôleur hepiaThanatosBundle:Network:index, hepiaThanatosBundle:Network correspondant au nom de notre contrôleur et index à la fonction qui est appelé. On remarque aussi qu'on passe une valeur par défaut à page.

##### Controller

Le contrôleur permet de relier la logique du site internet à la partie affichage. Quand une url est demandée par l’utilisateur, le routeur va appeler une fonction du contrôleur associé. Le contrôleur va ensuite s’occuper de traiter la demande en appelant les services et les modèles. Pour finalement renvoyer une réponse au client.

Dans le cas où l’utilisateur demande l’url /set/3, elle est associé au contrôleur hepiaThanatosBundle:Set:view donc la fonction view sera appelé dans le contrôleur Set.



Dans le code ci-dessus on peut voir que la fonction view récupère les données concernant le réseau grâce à doctrine qu’on détaillera plus tard. Elle s’occupe ensuite de renvoyer les données récupérées au client grâce à la fonction render en passant par une template twig qui sera aussi détaillé plus tard.

On peut voir qu’on récupère aussi directement les paramètres passer par l’url grâce au routeur qui va appeler la fonction directement avec l’id.

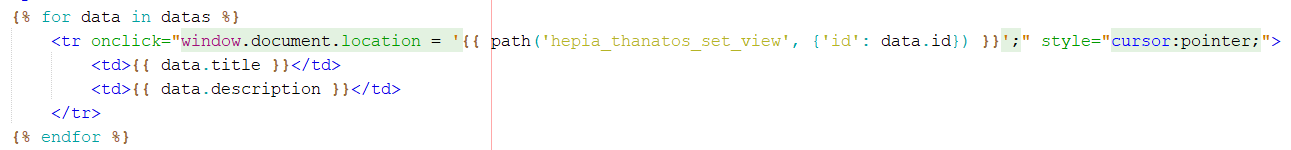
##### Twig

Twig est moteur de template qui nous permet de séparer le code PHP du code html. Il nous permet ainsi de gagner en visibilité.



Comme vous pouvez le voir dans le code ci-dessus on utilise la fonction render pour générer une template twig et ainsi lui passer les paramètres à afficher sur la page en question.

Twig possède une syntaxe simplifiée pour afficher les paramètres sur la page html.



Voici un exemple de code Twig et Html, on peut voir qu’on récupère le paramètre datas passer au préalable et qu’on itère sur ce paramètre pour afficher toute les datas. Pendant l’itération on peut accéder aux données de data qui sont le titre et la description.

Twig nous permet aussi d’étendre nos template dans le cas de mon client web j’utilise un template principale qui s’occupe de gérer le menu et la structure générale du site web et pour changer de page j’étends le template principal avec des sous templates qui me permettent d’afficher la page en question.

##### Doctrine

Doctrine est une couche d’abstraction à la base de données sur PHP. Il permet de relier nos objets php aux éléments de la base de données. C’est un outil très puissant qui automatise la création des tables et relations dans la base de données et permet au développeur de s’occuper seulement d’objet php sans se soucier des requêtes sqls derrières.

#### Docker

##### Dockerfile

Dockerfile permet à Docker d’automatiser la création d’une image Docker via des instructions. Pour se faire il suffit de créer un répertoire vide et de créer un fichier Dockerfile dedans ensuite la commande « *docker build* » suffira à générer l’image avec les instructions contenu dans le Dockerfile.

Les instructions sont exécutées dans l’ordre et sont indépendantes, chaque instruction va générer une nouvelle image et cette image sera utilisée par la suivante pour ajouter la nouvelle action.

Ce système permet d’avoir une génération rapide en cas de changement, par exemple si je décide de modifier la dernière instruction cela n’affectera pas les instructions précédentes et par conséquent seulement la dernière instruction sera exécutée.

Le format des instructions est le suivant :

INSTRUCTION arguments

Les principaux types d’instruction sont : FROM, RUN, CMD, COPY

FROM : Permet de définir une image de base comme le système d’exploitation que l’on souhaite utiliser, on peut aussi utiliser une de nos images déjà construite ou bien des images fournies par d’autre utilisateur de Docker avec des éléments déjà installés et configurés comme un serveur web par exemple. C’est une des grandes forces de Docker car nous avons à notre disposition énormément d’images avec des services déjà installés qu’on peut personnaliser à notre guise.

Format : « FROM <image> »

RUN : L’instruction run est la plus utilisée, elle permet de lancer des commandes sur l’image. Notamment des commandes d’installations via apt-get install sous une distribution linux. Evidemment elle ne se limite pas à des commandes d’installations elle peut aussi exécuter toutes les commandes disponibles sur le système d’exploitation utilisé.

Format : « RUN <command> »

CMD : CMD est très similaire à RUN mais contrairement à cette instruction, la commande n’est pas exécutée au moment de la construction de l’image mais au lancement de celle-ci cqui nous permet de démarrer nos différents services ou scripts. Du coup RUN est réservé pour la phase de l’installation et CMD pour la phase de l’exécution.

Format : « CMD command param1 param2 »

COPY : Cette instruction permet simplement de copier des données de la machine hôte à l’image Docker. Ce qui permet par exemple de copier un script ou un site web sur l’image.

Format : « COPY <src> … <dest> »

##### Container

Un container Docker est une machine lancée à partir d’une image, chaque container possède une identification unique. Une fois arrêté il peut être relancé via une commande. Il est possible de lancer plusieurs containers de la même image. Contrairement à une machine virtuelle le container ne simule pas les devices il utilise directement les ressources de la machine ce qui permet d’avoir de meilleures performances qu’une machine virtuelle. Une adresse IP est attribuée à chaque container, cette adresse IP est fournie directement par le sous-réseau créé par Docker et est accessible par l’ordinateur hôte. Pour accéder à ces machines depuis l’extérieur Docker utilise un système de redirection de port.

#### Websocket

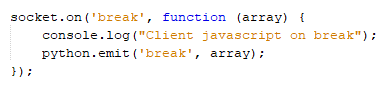
##### Socket.io

Socket.io est une bibliothèque qui permet de faire des communications synchrones dans une application. Socket.io est basé sur différentes techniques de communication en temps réel, notamment les websockets.

##### Messages

Dans socket.io nous avons la possibilité d’émettre des messages à tous les clients (broadcast) ou bien à un client spécifique.

Concernant la réception le serveur est en « écoute » sur un type de message et dès qu’il reçoit se message il va le traiter.



On peut voir dans le code ci-dessus que le serveur attend de recevoir le message break, une fois qu’il l’a reçu il va envoyer se message en broadcast sur les différents clients python.

##### Namespaces

Les namespaces dans socket.io sont très utiles, ils permettent d’assigner différent point d’arrivée ou chemin. C’est utile pour séparer les différentes parties de son application et minimiser les requêtes. Chaque namespace est un canal de communication différente. Par défaut tous les clients sont connecté sur le namespace « / ».

Dans le cas de mon travail j’ai utilisé les namespaces pour séparer les différents clients (php, javascript et python) ainsi chaque client a son propre canal de communication et peut communiquer avec les autres quand cela est nécessaire.

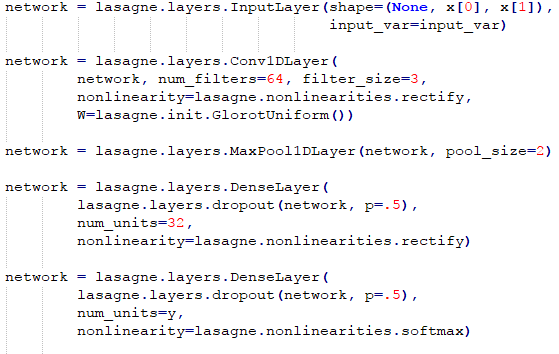
#### Lasagne

##### Layers

Les layers (couches) dans lasagne permettent de construire les différentes architectures de réseau de neurones convolutionnels.

Les principales couches disponibles sont couche entièrement connectées, couche convolutionnel et couche de perte.

Le fonctionnement de Lasagne est très simple pour créer son architecture il suffit d’empiler les différentes couches et de les relier avec les paramètres d’entrée et de sortie.



L’exemple ci-dessus représente un des réseaux que j’ai utilisé dans le cas de l’apprentissage des tweets positifs ou négatifs sur twitter. On peut voir que mon architecture comprends une convolution avec 64 filtres de taille 3 avec une couche de pooling qui va diviser la taille par 2 pour terminer sur une couche entièrement connectée de 32 unités.

## Analyse organique

### Architectures



L’architecture de mon projet est composée d’une base de données, d’un serveur Nodejs, d’un client python et d’un client web.

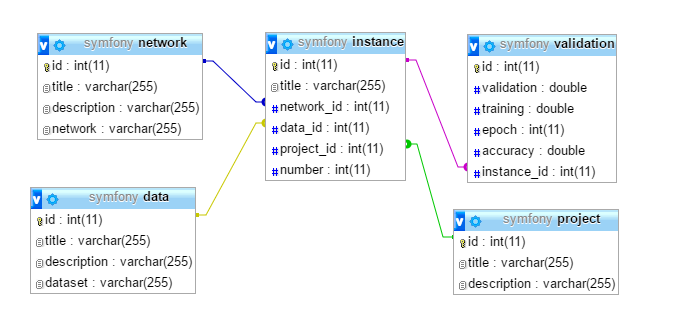
La base de données permet de stocker les différentes données concernant les réseaux de neurones.

Le serveur Nodejs s’occupe de la communication entre les différents clients via les Websockets.

Le client web permet à l’utilisateur d’avoir une application graphique et ainsi interagir avec le serveur Nodejs pour envoyer différentes requêtes.

Le client python est une instance docker qui contient un script python qui s’occupe de gérer la communication avec le serveur Nodejs et aussi l’entraînement d’un réseau de neurones convolutifs.

### Base de données



Les tables dans l’image précédente ont été générées grâce à Doctrine. Je vais détailler les différents champs et relations dans ce chapitre.

**Network :** La table Network permet de stocker les différentes architectures neuronales de l’application. Les champs sont title, description et network. Le champ title permet de donner un titre à l’architecture, généralement on reprend le nom du fichier concerné. Le champ description permet de décrire brièvement l’architecture par convention on écrit les différentes couches de l’architecture. Pour terminer le champ network permet de mettre en relation le fichier qui contient le code de l’architecture à la base de données. Cette table contient une relation avec la table instance, cette relation permet de définir qu’une instance est reliée à un seul Network mais qu’un Network peut être utilisé par des instances différentes.

**Data :** La table Data est fortement similaire à la table Network excepté qu’elle contient les ensembles de données à la place des architectures.

**Project :** La table Project permet de stocker les différents projets de l’application, un projet permet de grouper des instances sur la même expérience. Elle contient un champ title et description qui permettent à l’utilisateur de nommer et décrire le projet.

**Instance** **:** La table instance est la table principale de la base de données, elle contient les champs title et number. Les autres champs sont des relations avec les autres tables de la base de données. Le champ title permet simplement de nommer l’instance par convention on nomme les instances en fonction des titres du Network et Data, le champ number permet de définir le numéro de l’ensemble de données à utiliser, ce champ permet de gérer la validation croisée. Pour résumer les relations avec les autres tables, une instance appartient à un projet et utilise une seule architecture et un seul ensemble de données.

**Validation :** La table Validation, permet de stocker les différentes données retournées par les clients pythons. C’est cette table qui permet de générer les différents graphiques des instances. Elle contient les champs validation, accuracy, training, epoch. Le champ epoch permet de déterminer le numéro de l’itération du client, le champ training permet de déterminer la fonction de perte concernant l’apprentissage, le champ validation permet de stocker la fonction concernant la validation et pour terminer le champ accuracy permet de stocker le pourcentage de réussite sur l’ensemble des données de test. Cette table possède une relation avec la table instance car plusieurs validations appartiennent à une instance.

### Serveur

Le serveur Nodejs est le pilier du projet, il s’occupe de la communication entre les différents clients web et python. Il contient aussi les différentes architectures et ensemble de données ainsi que les sauvegardes.

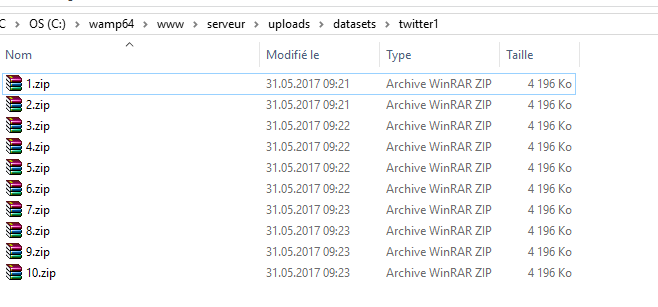
#### Les fichiers

Le serveur contient un répertoire uploads où sont stockés les différents fichiers nécessaires au bon fonctionnement de l’application.

Dans le cas de l’ajout d’une nouvelle architecture ou d’un nouvel ensemble de données il faudra suivre le format mis en place et décrit dans cette partie.

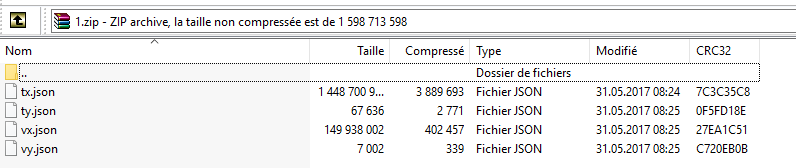
Le répertoire uploads contient les répertoires datasets, networks et saves.

Le répertoire datasets contient les différents ensembles de données sous le format suivant : un répertoire avec le nom de l’ensemble des données par exemple « twitter1 » ce répertoire doit contenir des fichiers zip de 1 à 10 qui correspondent aux différents ensembles de la validation croisée.



Chaque fichier zip doit contenir 4 fichiers : tx.json, ty.json, vx.json et vy.json.

* tx.json et ty.json correspondent aux entrées sorties de l’ensemble d’entraînement.
* vx.sjon et vy.json correspondent aux entrées sorties de l’ensemble de test.



Le répertoire networks contient les différentes architectures disponibles. Le nom est libre, la seule contrainte est que l’extension du fichier doit être .py. Le format du ficher sera décrit plus en détail dans la partie client python.



Concernant le répertoire saves il est géré automatiquement par le serveur. Il va stocker une sauvegarde par instance en fonction de l’id de l’instance. Ceci permet à l’utilisateur de télécharger le résultat de l’entraînement et aussi à l’instance de reprendre l’entrainement où il s’est arrêté.

#### Les routes

Le serveur met à la disposition du client, différentes routes pour télécharger les différents fichiers nécessaires au fonctionnement de l’entrainement du réseau de neurone comme l’architecture et l’ensemble de données. Certaines routes permettent aussi de récupérer des données ou d’en envoyer. Donc on peut dire qu’une partie de notre serveur fait office de serveur REST.

/networks : Cette route permet d’avoir la liste des différentes architectures disponibles. Le format de sortie est du type json. Cette route est utilisée par le client web pour mettre en relation un fichier avec un titre et une description dans la base de données.

/datasets : Très semblable à /network, il permet de récupérer la liste des différents ensembles de données disponibles.

/datasets/{dataset}/{number} : Cette route permet de récupérer un ensemble de données en téléchargement. {dataset} correspond au nom de l’ensemble de données et {number} correspond au numéro de la validation croisée. Cette route est utilisée par les clients python pour récupérer les ensembles de données au moment de l’entraînement.

/networks/{network} : Cette route est semblable à /datasets/{dataset}/{number} excepté qu’elle permet de récupérer l’architecture à la place de l’ensemble de données.

/saves/get/{id} : Cette route permet de récupérer les sauvegardes en fonction de l’id d’une instance. Elle est utilisée par les deux clients.

/saves/post/{id} : Cette route permet d’envoyer sur le serveur Nodejs la sauvegarde effectuée de l’entraînement d’une architecture en fonction de l’id d’une instance. Elle est utilisée par le client python.

#### Websocket

Le serveur utilise trois namespaces différents pour les clients : javascript, python et php. Pour rappel, les namespaces permettent de séparer les canaux de communication.

##### Réception

Les différents messages que le serveur attend venant d’un client javascript :

* Start : Permet de lancer une instance sur un des clients pythons. On doit passer l’id de l’instance et le nom du client python.
* Resume : Permet de reprendre une instance. On doit lui passer le nom du client python.
* Break : Permet de mettre en pause une instance. On doit lui passer le nom du client python.
* Stop : Permet d’arrêter une instance. On doit lui passer le nom du client python.
* Save : Permet de sauvegarder une instance. On doit lui passer le nom du client python

Les différents messages que le serveur attend en provenance d’un client python :

* Validation : Permet de recevoir les résultats de l’apprentissage de la part du client python. On doit lui passer la validation, l’entrainement, l’itération et la précision en paramètre.
* Data : Permet de mettre à jour les différentes informations contenues dans le client python notamment l’état de celui-ci. On doit lui passer en paramètre le nom, le statut, l’erreur et l’état du client python.
* Disconnect : Ce message permet de détecter quand le client se déconnecte. Cela permet de savoir si l’instance du client python a eu un problème.

Les différents messages que le serveur attend en provenance d’un client php :

* Available : Permet de récupérer la liste des clients pythons disponibles pour lancer une instance.
* Launch : Permet de lancer une instance sur un des clients pythons. On doit passer l’id de l’instance et le nom du client python.

##### Émission

Les différents messages que le serveur peut émettre aux différents clients :

* Computers [Serveur -> Client javascript] : Permet de recevoir la liste des différentes instances avec leurs paramètres.
* Start [Serveur -> Client python] : Permet de transmettre le message start au client python.
* Resume [Serveur -> Client python] : Permet de transmettre le message resume au client python.
* Break [Serveur -> Client python] : Permet de transmettre le message break au client python.
* Stop [Serveur -> Client python] : Permet de transmettre le message stop au client python.
* Save [Serveur -> Client python] : Permet de transmettre le message save au client python.
* Validation [Serveur -> Client javascript] : Permet de recevoir en temps réel les validations.
* Available [Serveur -> Client php] : La réponse pour la liste des clients disponibles.

#### Fonctionnement

Le serveur contient une liste des clients pythons avec les informations suivantes :

* Name : Le nom fait d’office d’identification pour chaque client python.
* Connected : Permet de savoir si le client python est actuellement connecté ou déconnecté.
* Status : Permet de connaître le statut du client python pour savoir ce qu’il est en train de faire.
* Error : Permet de savoir si une erreur est survenue et à quelle étape.
* State : Permet de connaître l’état du client python. La machine d’état sera détaillée plus tard.
* Id\_project : Permet de connaître l’id du projet qui a lancé une instance.
* Title : Permet de connaître le titre d’une instance.

Lorsque qu’un client python se connecte pour la première fois au serveur Nodejs. Il va transmettre ses informations notamment son nom. Le serveur Nodejs va l’ajouter dans une liste et va s’occuper de transmettre les informations nécessaires aux clients web. Dans le cas d’une déconnection du client python et d’une reconnexion, le serveur Nodejs va reconnaître ce client python grâce à son nom. C’est pourquoi il est nécessaire que chaque client python possède un nom différent, dans le cas contraire ça pourrait entrainer des comportements impondérables.

Concernant la communication des ordres du client web au client python. Le serveur va juste s’occuper de transmettre en broadcast sur les clients pythons l’ordre du client web. Tous les clients pythons vont recevoir cet ordre mais seulement le client qui reconnait son nom va l’exécuter. J’ai choisis le système de broadcast pour pouvoir déployer plus rapidement sans avoir à rechercher à chaque fois le socket correspondant dans la liste.

### Client

Comme expliqué dans les chapitres précédant, le projet possède deux types de clients, un client web qui permet d’envoyer des ordres et afficher les résultats. Et un client python qui reçoit les ordres et s’occupe de gérer l’entraînement du réseau de neurones.

#### Python

##### Fonctionnement

Les différentes communications possibles avec le client python ont déjà été détaillées dans la partie du serveur. Par conséquent ce chapitre sera consacré au fonctionnement du client python.



La première étape du client python est de se connecter au serveur Nodejs. Une fois connecté il va se mettre dans un état d’attente passive. C’est au moment de la connexion que le client envoi ses informations.

Le client va attendre un ordre de lancement pour passer de l’état d’entraînement. Cet ordre et ainsi que les autres ordres possibles sont envoyés par le serveur Nodejs.

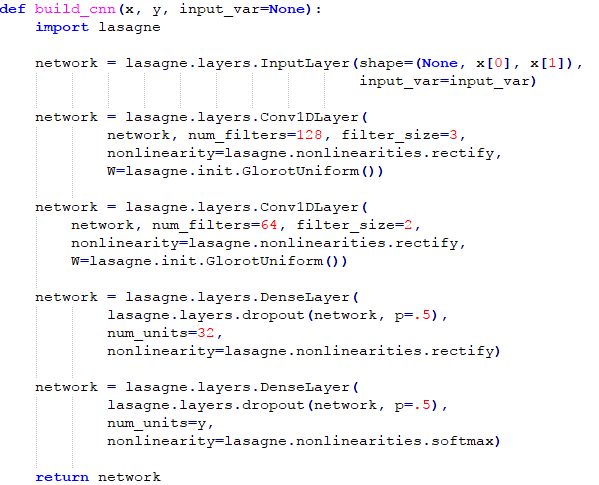
Quand l’entraînement est lancé le serveur va premièrement télécharger les ensembles de données. Ensuite il va télécharger et exécuter l’architecture. Une fois cette étape terminé le client va commencer l’entraînement de notre réseau de neurones via l’ensemble de données et l’architecture téléchargé.

Pendant cette phase d’entraînement il est possible de recevoir trois ordres : pause, stop et sauvegarde.

Pause : La pause permet de mettre en pause l’entraînement, une fois dans cette état une attente passive est activée. Pour sortir de cet état on attend l’ordre « resume » qui permet de retourner dans l’état d’entraînement.

Stop : Permet de stopper l’entraînement, cet ordre permet de retourner à l’état d’attente. Comme indiqué dans l’image en cas d’erreur ou d’interruption système ce changement d’état est aussi effectué. Cela permet d’arrêter correctement un entraînement et ainsi évité des problèmes dus à l’interruption inopiné du programme.

##### Sauvegarde : Permet de sauvegarder l’entraînement d’un réseau de neurones. Une fois l’action effectué on retourne directement à l’entraînement.Architecture



Pour la création de l’architecture, on reprend exactement le code de lasagne. Par conséquent l’utilisateur est libre d’utiliser toute les options proposées par lasagne en évitant de devoir coder plus que nécessaire.

Comme indiqué dans le chapitre de lasagne on construit l’architecture couche par couche. Les seules différences notables ici c’est qu’on est obligé d’utiliser l’entête de fonction montré dans le code d’exemple. L’import lasagne est aussi obligatoire et le return network. On remarque aussi que dans la première et dernière ligne on utilise nos paramètres.

X correspond aux entrées de notre réseau de neurones et y aux sorties.

Input\_var permet de déterminer la dimension de notre architecture par exemple si on est en 2d ou 3d.

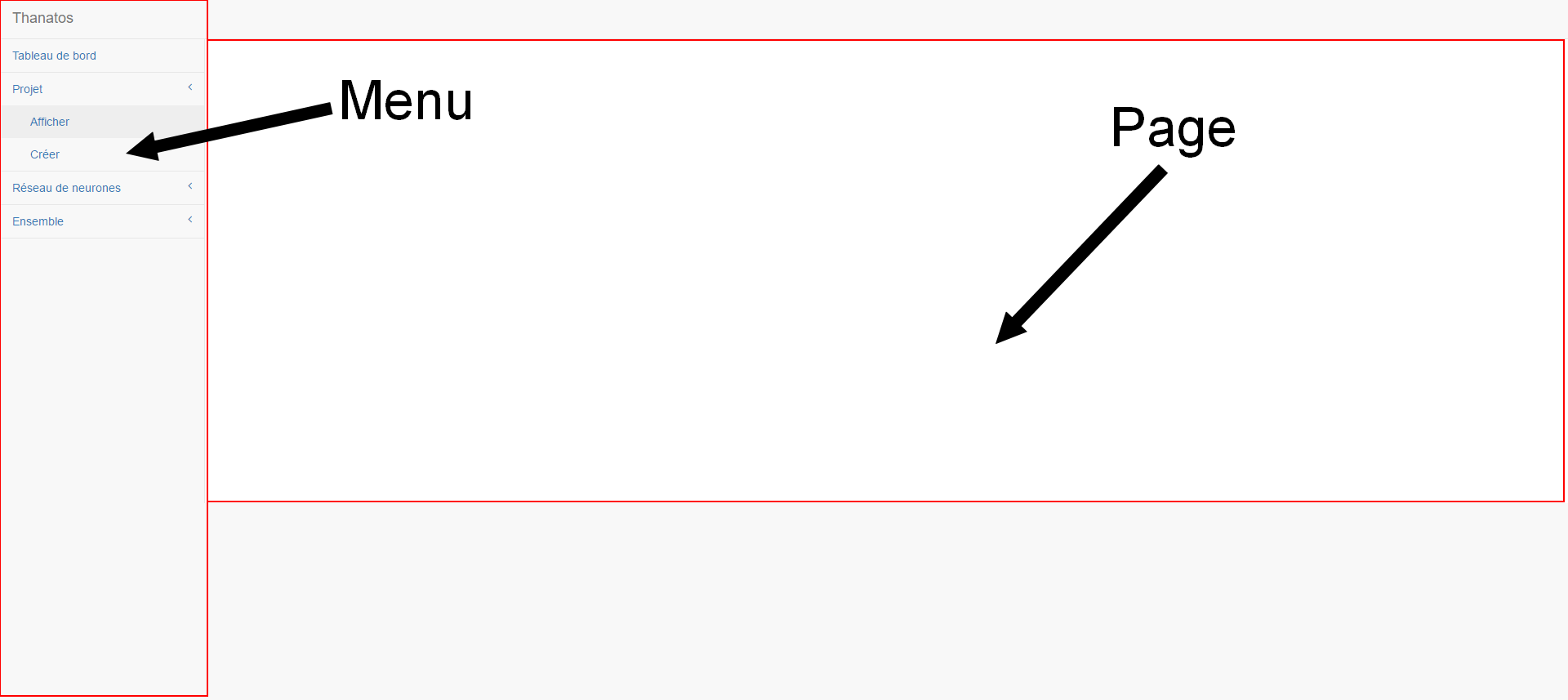
Ce code est ensuite chargé en mémoire et exécuté par notre client python. Si jamais ce code contient une erreur elle sera indiquée par le client python et l’entraînement s’arrêtera.

#### Web

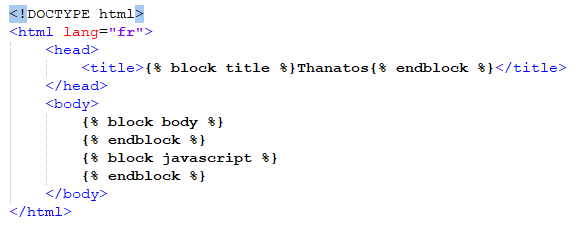
##### Modèle

##### Vue

Concernant la vue, j’ai utilisé twig une des fonctionnalités de symfony qui permet de séparer le code html du code php en utilisant un parser.

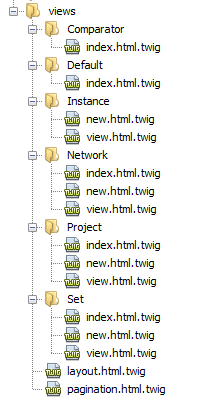


Twig permet de créer une template avec des éléments permanant et des blocs modifiables. Dans notre cas le menu reste permanant et la page va changer. La structure de ma template ressemble au code suivant :



Le block title permet de changer le titre de la page. Le block body permet de changer le contenu de la page. Le block javascript permet de rajouter simplement du javascript en fonction de la page.

Le menu est contenu directement dans le code vu précédant. Seulement les codes contenus dans les blocks seront modifiés en fonction des pages.



Dans l’image précédente, nous pouvons voir l’architecture des répertoires concernant l’affichage de notre site web. Les répertoires sont nommés en fonction des contrôleurs qui va les utiliser et les noms des fichiers twig sont nommées par rapport à la fonction qui va faire appel l à eux. Layout représente la template général du site web que nous avons vu précédemment et pagination permet d’inclure l’affichage de la pagination sans récrire le code html qui lui est propre.

##### Contrôleur

###### Set, Project et Network

Set, Project et Network sont des contrôleurs au fonctionnement identique. Nous allons détailler leur fonctionnement ici.

Ces trois contrôleurs permettent à l’utilisateur de voir, créer et afficher la liste des projets, des ensembles de données et des architectures.Ils contiennent trois fonctions : indexAction, newAction et viewAction.

indexAction : Cette fonction permet d’afficher la liste des différentes données par exemple la liste des projets. Un système de pagination a été créé du coup elle récupère les données en fonction de la page et les affiches dans la vue appropriés.

newAction :

## Tests et performances

### Tests

|  |  |  |
| --- | --- | --- |
| Test | Résultat attendu | Fonctionnement correct |
| Créer un nouvel ensemble de données |  |  |
| Créer une nouvelle architecture |  |  |
| Créer une nouvelle instance |  |  |
| Créer une nouvelle instance avec validation croisée |  |  |
| Démarrer une instance |  |  |
| Arrêter une instance |  |  |
| Mettre en pause une instance |  |  |
| Sauvegarder une instance |  |  |
| Mettre une erreur dans l’architecture |  |  |
| Mettre une erreur dans l’ensemble de données |  |  |
| Arrêter un pc avec une instance python client en cours |  |  |
| Arrêter le serveur Nodejs avec des instances en cours |  |  |
| Télécharger une sauvegarde |  |  |
| Exécuter le résultat d’une sauvegarde |  |  |

### Performances

Matériel utilisé pour les tests de performances :

* Intel core i7 (CPU)
* GeForce GTX 860M (GPU)
* 16G RAM

Les tests ont été effectué avec l’architecture et l’ensemble de données MNIST. Le code utilisé contient seulement la partie d’entrainement et ne gère pas la connexion au serveur Nodejs.

|  |  |
| --- | --- |
| Système d’exploitation | Ubuntu 16.04 |
| Unité de calcul | CPU |
| Temps (Moyenne de 20 itérations) | 10.30905263 s |
| Docker | Non |

|  |  |
| --- | --- |
| Système d’exploitation | Ubuntu 14.04 |
| Unité de calcul | CPU |
| Temps (Moyenne de 20 itérations) | 10.46694737 s |
| Docker | Oui |

Comme nous pouvons le voir sur les premières mesures docker n’influence quasiment pas le temps d’apprentissage. On suppose que la différence entre les deux temps serait semblable avec une moyenne plus élevée.

|  |  |
| --- | --- |
| Système d’exploitation | Ubuntu 16.04 |
| Unité de calcul | GPU |
| Temps (Moyenne de 20 itérations) | 1.361684211 s |
| Docker | Non |

|  |  |
| --- | --- |
| Système d’exploitation | Ubuntu 14.04 |
| Unité de calcul | GPU |
| Temps (Moyenne de 20 itérations) | 1.414315789 |
| Docker | Oui |

Pour docker on remarque la même similitude que pour les GPUS. C’est surtout le gain de temps par rapport au CPU qui est intéressant dans cette deuxième mesure. Nous sommes presque 10x plus performant que dans le premier cas.

## Déploiement

## Installations

## Amélioration

### Gestion ssh

Une gestion ssh des instances simplifiées par l’interface web pourrait être intéressante pour redémarrer un container docker à distance. Cela permettra à l’utilisateur d’être complétement autonomes par rapport à l’instance.

### Déploiement intelligent

Rajouter une couche d’abstraction pour l’utilisateur qui n’aura plus la possibilité de choisir l’ordinateur pendant la phase de création d’instance. Le serveur Nodejs s’occupera de choisir la meilleure machine pour l’entrainement en fonction de l’architecture et l’ensemble de données. On pourrait même penser à un système de migration dynamique entre les machines pour optimiser la répartition des charges quand nécessaire.

### Interruption du serveur

Actuellement quand le serveur Nodejs s’arrête les instances en cours sont perdues. Il serait intéressant de mettre en place un système de récupération pour éviter la perte des instances.

### Sauvegarde automatisée

Pour l’instant l’utilisateur peut seulement sauvegarder manuellement, une possibilité serait de programmer des sauvegardes automatiques à intervalle régulier pour éviter de perdre notre entraînement lors d’une coupure.

### Comparateur de graphique

Actuellement le comparateur de graphique permet juste de comparer plusieurs graphiques entres eux avec une moyenne de ces graphiques. Il serait intéressant de rajouter la possibilité de groupé des graphiques et comparer leurs moyennes.

### Configuration des fonctions d’apprentissages

La possibilité à l’utilisateur de gérer ses propres fonctions d’apprentissage un peu sous la même forme que l’architecture. Ainsi l’utilisateur pourra expérimenter plus facilement diverse possibilité sans changer le code du client python.

### Utilisation du format ndjson

Actuellement le format json est utilisé pour stocker et charger les ensembles de données. Le format ndjson ressemble fortement à json mais il comporte l’avantage d’être plus léger ce qui permettrait de diviser par deux la taille des ensembles de données.

# Expérience

## Introduction

Dans cette troisième partie du travail de bachelor, je vais mener une expérience avec l’application que j’ai créée dans la deuxième partie. Cela va me permettre de tester le bon fonctionnement de cette application et aussi d’expérimenter les réseaux de neurones.

# Références

<https://en.wikipedia.org/wiki/Deep_learning>

https://fr.wikipedia.org/wiki/R%C3%A9seau\_neuronal\_convolutif

https://fr.wikipedia.org/wiki/Apprentissage\_non\_supervis%C3%A9

https://fr.wikipedia.org/wiki/Convolution\_de\_Dirichlet#D.C3.A9finition.2C\_exemples\_et\_premi.C3.A8res\_propri.C3.A9t.C3.A9s

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

http://cs231n.github.io/convolutional-networks/

<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>

<https://fr.wikipedia.org/wiki/Word_embedding>

<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

<https://docs.docker.com/>

https://nodejs.org/en/docs/